Vue.js

Intro

Vue是一套用于构建用户界面的**渐进式框架**。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层

HelloWorld例程

```
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8"><title>Vue Todo Tutorial</title></head>
<body>
<div id="app">{{ message }}</div>
</body>
<script src="https://cdn.bootcss.com/vue/2.5.16/vue.js"></script>
<!--可选连接：-->
<!--<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>-->
<!--<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.1.8/vue.min.js">
</script>-->
<!--<script src="https://unpkg.com/vue/dist/vue.js"></script>-->
<script>
    var app = new Vue({
        el: '#app',
        data: function () {
            return {message: 'Hello Vue!'}
        }
    })
</script>
</html>
```

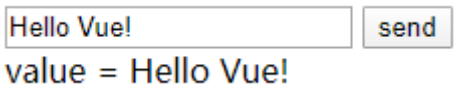
我们可以在 Vue.js 的官网上直接下载 vue.min.js 并用 标签引入：https://vuejs.org/js/vue.min.js。

在上面的例程中我们创建了一个`Vue`对象，然后指定其挂载的元素`el`（elements）是id为“app“的那个`div`块，并且在`data`中绑定了一个叫`message`的变量并赋值。

表单绑定

```
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8"><title>Vue Todo Tutorial</title></head>
<body>
<div id="app">
    <input type="text" v-model="value">
    <input type="button" value="send">
    <div>value = {{value}}</div>
</div>
</body>
<script src="https://cdn.bootcss.com/vue/2.5.16/vue.js"></script>
<script>
    var app = new Vue({
```

```
        el: '#app',
        data: function () {
            return {value: 'Hello Vue!'}
        }
    })
</script>
</html>
```



这里`input`标签中的`v-model`是Vue的指令之一，用于将`input元素value属性的值`和我们创建的`Vue对象中value的值`进行绑定。经过这样的绑定之后，input框中引起的value值变化就会**实时**地反映到关联的Vue对象中，所以下方的{{value}}也会跟随上方输入框**实时**变化。

网页通知

我们可以在Vue对象中增加一些函数的方法，比如如下这个可以产生网页通知的方法：

```
<input type="button" value="send" v-on:click="send"/> <!--增加按键动作-->
<!---->
<script>
var app = new Vue({
        el: '#app',
        data: function () {
            return {message: ''}
        },
        methods:{                           <!--定义方法-->
            send: function () {
                alert("send successful");
                this.message=''               <!--清空Vue/输入框内容-->
            }
        }
    })
</script>
```

↑这里的`methods`中可以添加多个方法。

实时统计输入的字数

Vue 还可以根据绑定的数据做一些计算，然后我们就可以引用计算的结果：

```
<div>count = {{count}}</div>                    <!--这里引用count这个键-->
<script>
    var app = new Vue({
        el: '#app',
        data: function () {
            return {message: ''}
        },
        <!--计算属性申明到 computed 对象里，这个对象的键是计算的结果，值是计算函数-->
        computed:{
            count:function () {
                return this.message.length;      <!--这里返回message的长度-->
            },
```

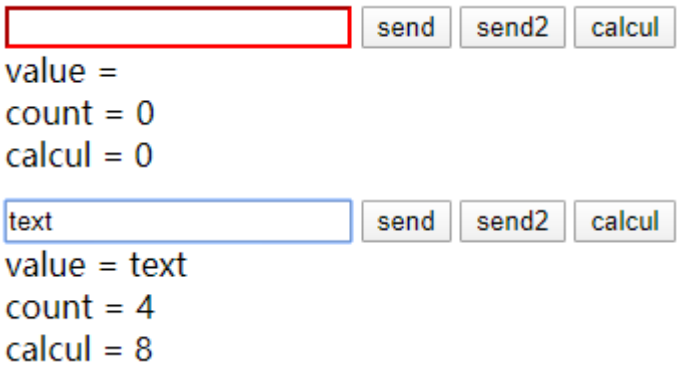
```
            calcul:function () {
                return this.count *2;            <!--这里引用了count这个键，返回其两倍的值-->
            }
        }
    })
</script>
```

绑定CSS样式

我们希望 input 中没有任何值输入，即 value 的值为空时，input 的边框为红色以提醒用户没有内容：

```
<style>
    .empty{
        border-color: red;
    }
</style>
<input type="text" v-model="message" v-bind:class="{empty: !count}">
```

这里使用了 `v-bind` 指令进行绑定，Vue会根据 `empty` 后的表达式 `!count` 的真假来判断 `class` 的值是否为 `empty`。



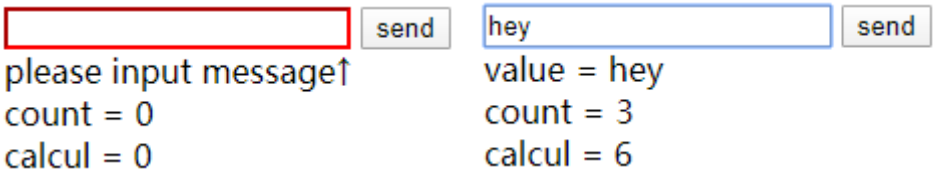
使用v-if进行判断

希望只有用户真正地输入了内容后，才提示 value 的值，否则显示请输入值：

```
<div id="app"> <!--无关内容已去除-->
    <input type="text" v-model="message" v-bind:class="{empty: !count}">
    <div v-if="notnull">value = {{message}}</div>
    <div v-else>please input message↑</div>
</div>
<script>
    var app = new Vue({
        el: '#app',
        data: function () {
            return {message: ''}
        },
        computed:{
            notnull:function () {
                return this.message!=='';
            }
        }
    })
</script>
```

这里的`<div v-if="notnull">value = {{message}}</div>`是重新定义了一个用于判断非空的函数，其实也可以直接用`<div v-if="count">value = {{message}}</div>`来判断，因为当未输入时count的值为0，即false。

这里的`<div v-else>please input message↑</div>`则是和上面搭配使用的。若没有这句代码的话就这行啥都不会显示。



v-for循环

假设我们人为构造数据如下：

```
<script>
    var app = new Vue({
        el: '#todo-app',
        data: function () {
            return{
                todos:[
                    {id:0, title:"learn chinese"},
                    {id:1, title:"learn english"},
                    {id:2, title:"learn german"}
                ]
            }
        }
    })
</script>
```

则可以用如下的方式进行显示：

```
<div id="todo-app">
    <div>
        <ul>
            <li v-for="todo in todos" :key="todo.id">  <!--这里使用的唯一的key为了高效的更新虚拟DOM-->
                <span>{{todo.title}}</span>
                <input type="button" value="completed">
                <input type="button" value="delete">
                <input type="text" placeholder="edit todo">
            </li>
        </ul>
    </div>
</div>
```

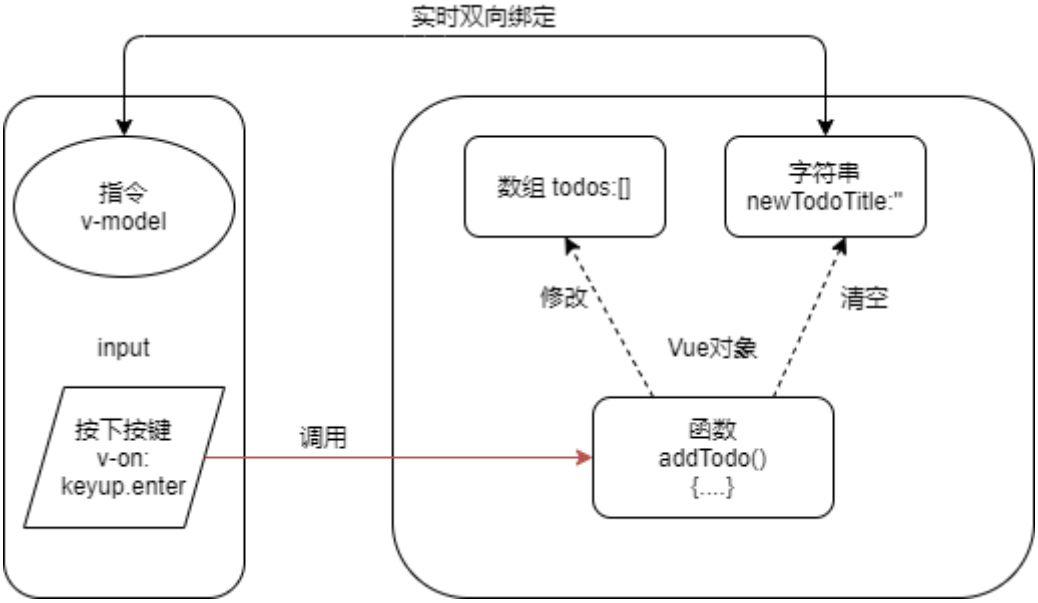
添加TODO

为了知道用户输入了什么内容，我们使用`v-model`指令将`input 的 value 值`和`Vue 的实例`绑定，这样在 Vue 中我们就知道了用户输入的值。

在监听键盘事件时，我们经常需要检查详细的按键。Vue 允许为 `v-on` 在监听键盘事件时添加按键修饰符：下面的`v-on:keyup.enter="addTodo"`说明当`key`是`enter`时调用`addTodo`函数。同理也可以为其他键，如`v-on:keyup.page-down`或者`v-on:keyup` -- `.tab` `.delete` `.esc` `.space` `.up` `.down` `.left` `.right` 。

```
![vue](D:\浏览器下载\vue.png)<div>
        <input type="text" placeholder="add todo" v-model="newTodoTitle" v-
on:keyup.enter="addTodo">
</div>
<!--------------------------------------------------------------------------
------------>
<script>
    let Id = 0;
    var app = new Vue({
        el: '#todo-app',
        data: function () {
            return{
                todos:[],
                newTodoTitle: ''
            }
        },
        methods:{
            addTodo: function () {
                if (this.newTodoTitle.trim()!==''){          <!--这里用来判断输入的
是否为纯空字符串-->
                    this.todos.push({id: Id++, title: this.newTodoTitle});
<!--塞入字符串数组-->
                    this.newTodoTitle = '';                                  <!--
清空字符串-->
                }
            }
        }
    })
</script>
```

基本关系可以总结成如下图↓，由于`v-model`指令的存在，只要在input框中输入，就会实时绑定数据到`newTodoTitle`这个字符串上，按下回车键就会调用`addTodo()`函数，在该函数中将`newTodoTitle`字符串塞入到`todos`数组中，并清空`newTodoTitle`字符串以便于在`input`框中继续输入新数据。



标记完成

1. 需要增加一个布尔值字段completed用于标记，初始化时为false，如这样：

```
addTodo: function () {
    if (this.newTodoTitle.trim()!==''){
        this.todos.push({id: Id++, title: this.newTodoTitle, completed: false});
        this.newTodoTitle = '';
    }
},
```

设置一个新的函数`markAsCompleted`用来标记该布尔值为true，即已完成：

```
markAsCompleted: function (todo) {    <!--这里的todo参数直接传就可以了-->
    todo.completed = true;           <!--修改completed这个布尔值-->
}
```

在按键`<input type="button" value="completed">`处调用上面这个函数，设置为已读，所以这个按键修改成这样：

```
<input type="button" value="completed" v-on:click="markAsCompleted(todo)">
```

2. 在显示部分，也需要在视觉上标记为已经完成的状态，这里用到定义的CSS格式：

```
<style>
    .completed {
        text-decoration: line-through;
    }
</style>
```

接下来修改一下，使得要显示的内容`<span>{{todo.title}}</span>`可以***动态绑定***上述样式，使用到的是`v-bind:class=""`的方式：

```
<span v-bind:class="{completed: todo.completed}">{{todo.title}}</span>
```

3. 这样基本的标记完成功能已经搞定，为了更加美观：我们在点击“完成”按钮后，该按钮消失，则可以小改一下，增加一个判断条件即可：“当该TODO未完成时则按钮存在”↓

```
<input type="button" value="completed" v-on:click="markAsCompleted(todo)" v-
if="!todo.completed">
```

4. 同时我们可以提供一个反悔的机会，即将已完成的任务重新设置为未完成，操作雷同，需要先增加一个设置为未完成的按钮，在该按钮中提供调用设置未完成的函数选项：

```
<input type="button" value="Not completed" v-on:click="markAsNotCompleted(todo)"
v-if="todo.completed">
```

当然也要提供这个方法了：

```
markAsNotCompleted: function (todo) {        <!--这里的todo参数直接传就可以了-->
    todo.completed = false;                 <!--修改completed这个布尔值-->
}
```

至于显示部分则不需要修改了，因为其是根据completed这个布尔值而进行动态绑定样式的。

删除TODO

经过上面的学习，可以雷同地进行一些操作，比如在delete这个按钮处放置函数入口：

```
<input type="button" value="delete" v-on:click="deleteTodo(todo)">
```

对于的函数则是这样：

```
deleteTodo: function (todo){
    this.todos.splice(this.todos.indexOf(todo), 1);
}
```

这里用到两个 JavaScript 数组的相关方法，this.todos.indexOf(todo) 用来定位元素的位置，然后我们删除掉这个位置的元素 splice(index, 1)，1 表示只删除一个，即当前位置（index 的值）的元素。

javascript 有一个 `confirm` 方法，调用该方法浏览器会弹出一个确认框，用户点击确认该方法会返回 true，点取消则不返回 true，因此用该方法可以实现等待用户确认删除的功能。

```
deleteTodo: function (todo){
    if (!confirm("Delete this TODO?")) {
        return
    }
    this.todos.splice(this.todos.indexOf(todo), 1);
}
```

编辑TODO

这里的思路是搞一个 `启用/禁用编辑功能` 的按钮，然后当编辑功能启用之后，会有一个多出来的input框用于修改内容。上述的按钮实现如下：

```
<input type="button" value="edit" v-on:click="pressEditButton(todo)">
```

对应的函数是：

```
pressEditButton: function (todo){
    todo.buttonEdit = !todo.buttonEdit;  //翻转状态
    this.newTodoTitle = '';              //用于后续操作中途放弃修改的时候情况缓存Todo
}
```

显然这里需要在存储的数据中添加一项布尔值 `buttonEdit`，原来的 `addTodo` 函数顺便修改为：

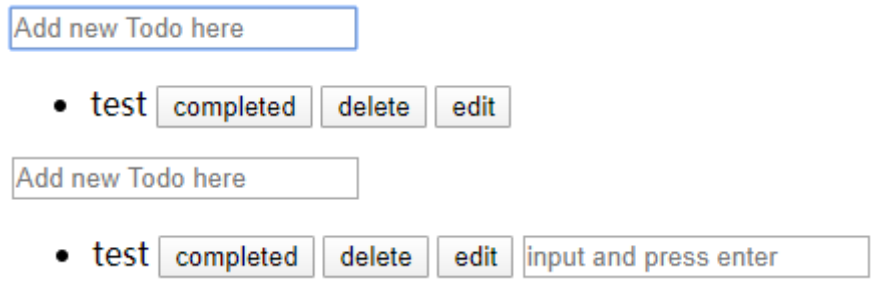
```
addTodo: function () {
    if (this.newTodoTitle.trim()!==''){
        this.todos.push({id: Id++, title: this.newTodoTitle, completed: false,
buttonEdit: false});
        this.newTodoTitle = '';
    }
},
```

而上面提到的input框则是这样：

```
<input type="text" placeholder="input and press enter"
       v-if="todo.buttonEdit"                //当修改按钮按下后才显示
       v-model="newTodoTitle"                //与newTodoTitle进行绑定
       v-on:keyup.enter="editTodo(todo)">     //输入后回车进入下面的函数
```

用于修改的函数为：

```
editTodo: function (todo) {
    if (this.newTodoTitle.trim() !== '') {                //判断非空
        this.todos.splice(this.todos.indexOf(todo), 1);  //清楚原来的内容
        this.todos.push({id: todo.id, title: this.newTodoTitle, completed:
false, buttonEdit: false});                                                    //保持原
来的id，内容更新
        this.newTodoTitle = '';                          //清空缓存
    }
}
```



自定义指令实现自动聚焦

我们希望用户在点击编辑按钮后光标自动聚焦在输入框中，而不是手动去点击一下再输入。input元素中有一个`focus`方法可以帮助实现这个功能，问题是如何在Vue中获得这个input元素？--> Vue的自定义指令可以实现。这里说的**指令**就是**用来指导被绑定的元素的行为**，比如之前接触到的`v-if` `v-model` `v-bind`。

我们可以按照类似的方法，实现一个focus指令，这个指令实现了inserted钩子函数，这个函数在被绑定的元素被插入dom时触发，且被绑定的元素会作为参数传入钩子函数，我们就可以在函数中对它操作。

```
<script>
    let id = 0; // 用于 id 生成
    var app = new Vue({
        ...
        methods: {
            ...
        },
```

```
        directives: {
            focus: {
                inserted: function (el) {
                    el.focus()
                }
            }
        }
    })
</script>
```

同时我们需要在input框中修改一下，添加这个绑定指令：

```
<input type="text" placeholder="input and press enter"
       v-if="todo.buttonEdit"
       v-model="newTodoTitle"
       v-on:keyup.enter="editTodo(todo)"
       v-focus="true">          <!--这里始终绑定，因为一旦出现就是为了修改TODO-->
```

全部标记为完成

逻辑很清晰，首先我们需要一个`全部完成`的按钮：

```
<input type="button" value="All Done" v-on:click="markAllAsCompleted">
```

只需要将`todos`数组中的所有项都过一遍`markAsCompleted`函数即可：

```
markAllAsCompleted: function () {
    this.todos.forEach((item) => {
        this.markAsCompleted(item)
    });
}
```

这里使用到了foreach语法，基本语法结构如上所示。

我们也可以使用`map`语法：

```
markAllAsCompleted: function () {
    this.todos.map(function (item) {
        item.completed = true;      //换了一个写法而已，
==this.markAsCompleted(item)
    })
}
```

计算未完成项数目

根据 Vue 对象已有的数据来计算新的结果，就是`计算属性`典型的应用场景。

```
<script>
    var app = new Vue({
        ...
        computed: {
            numNotCompleted: function () {
                return this.todos.filter(todo => !todo.completed).length
```

```
                //这里的检测函数是 todo=>!todo.completed
                //等价于：
                // function(todo){
                //  return !todo.completed;
                //}
            }
        }
    })
</script>
```

这里使用的是一个todo里面的completed属性进行过滤，filter是函数式编程的思想，顾名思义就是要**根据某个检测函数去列表中筛选出符合检测要求的结果**，会**返回所有检测为true的元素组成的列表**。同时我们需要添加显示项：

```
<span>{{numNotCompleted}} items left</span>
```

部分删除或全部删除

对应的两个选项是：

```
<input type="button" value="delete completed" v-on:click="deleteCompleted">
<input type="button" value="delete all" v-on:click="deleteAll">
```

关联的两个函数分别为：

```
deleteCompleted: function () {
    if (confirm("Delete all completed TODOs?")) {
        this.todos = this.todos.filter(todo => !todo.completed)
    }
},
deleteAll: function () {
    if (confirm("Delete all TODOs?")) {
        this.todos.forEach((item)=>{
            this.todos = [];
        })
    }
}
```

部分显示

可以想到这里需要有3个状态：所有`all`，正在做的`doing`，完成的`completed`。因此我们在显示区域需要有一个变量进行判断是显示啥，所以我们需要在Vue实例中增加一个变量`show`进行标识，并初始化为`all`，表示在默认状态下显示所有内容。

```
<input type="button" value="show doing" v-on:click="showDoing">
<input type="button" value="show completed" v-on:click="showCompleted">
<input type="button" value="show all" v-on:click="showAll">
```

而在上面的三个按钮分别绑定的函数就是用来修改这个变量的：

```
<script>
var app = new Vue({
```

```
    el: '#todo-app',
    data: function () {
        return{
            todos:[],
            newTodoTitle: '',
            show: 'all'              /*这里定义标识显示内容变量*/
        }
    },
    methods:{
        showCompleted: function () {
            this.show = 'completed'      /*设置显示已完成*/
        },
        showDoing: function () {
            this.show = 'doing'          /*设置部分正在做*/
        },
        showAll: function () {
            this.show = 'all'            /*设置全部显示*/
        }
    }
</script>
```

那么我们还需要一个函数，**根据当前show的状态用来返回待显示的数组**：

```
computed:{
    filteredShowTodos: function () {
        if (this.show === 'completed'){
            return this.todos.filter(todo => todo.completed);        /*返回已完成的
TODO*/
        }else if (this.show ==='doing'){
            return this.todos.filter(todo => !todo.completed);       /*返回正在做的
TODO*/
        }else {
            return this.todos;                                      /*返回全部TODO*/
        }
    }
},
```

那么在显示的那一块，就需要小改一下：

```
<ul>
            <li v-for="todo in filteredShowTodos" :key="todo.id">   <!--这里修改
了-->
                <span v-bind:class="{completed: todo.completed}">{{todo.title}}
</span>
                <input type="button" value="completed"
                       v-on:click="markAsCompleted(todo)"
                       v-if="!todo.completed">
                <input type="button" value="Not completed"
                       v-on:click="markAsNotCompleted(todo)"
                       v-if="todo.completed">
                <input type="button" value="delete"
                       v-on:click="deleteTodo(todo)">
                <input type="button" value="edit" v-
on:click="pressEditButton(todo)">
                <input type="text" placeholder="input and press enter"
                       v-if="todo.buttonEdit"
```

```
                        v-model="newTodoTitle"
                        v-on:keyup.enter="editTodo(todo)"
                        v-focus="true">
            </li>
</ul>
```

到此，部分显示功能已完成，我们还可以自定义一下其样式，使得被选中的显示框变红：

```
<input type="button" value="show doing"
       v-on:click="showDoing"
       v-bind:class="{selected:show===='doing'}">    <!--  因为这里是判断相等，所以使用===而不是：  -->
<input type="button" value="show completed"
       v-on:click="showCompleted"
       v-bind:class="{selected:show===='completed'}">
<input type="button" value="show all"
       v-on:click="showAll"
       v-bind:class="{selected:show===='all'}">
```

代码：

```
<!DOCTYPE html>
<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml"
      xmlns:v-on="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Vue Todo UI</title>
    <style>
        .completed {
            text-decoration: line-through;
        }
        .selected {
            color: red;
        }
        .inputting {
            color: red;
        }
    </style>
</head>
<body>

<div id="todo-app">
    <div>
        Please add new TODOs in the box below and press enter.
        <br>
        <input type="text" placeholder="Add here" v-model="newTodoTitle" v-on:keyup.enter="addTodo">
    </div>

    <div>
        <ul>
            <li v-for="todo in filteredShowTodos" :key="todo.id">
                <span v-bind:class="{completed: todo.completed}">{{todo.title}}</span>
                <input type="button" value="completed"
```

```
                        v-on:click="markAsCompleted(todo)"
                        v-if="!todo.completed">
                <input type="button" value="Not completed"
                        v-on:click="markAsNotCompleted(todo)"
                        v-if="todo.completed">
                <input type="button" value="delete"
                        v-on:click="deleteTodo(todo)">
                <input type="button" value="edit" v-
on:click="pressEditButton(todo)">
                <input type="text" placeholder="input and press enter"
                        v-if="todo.buttonEdit"
                        v-model="newTodoTitle"
                        v-on:keyup.enter="editTodo(todo)"
                        v-focus="true">
            </li>
        </ul>
    </div>

    <div>
        <span>Totally {{numNotCompleted}} items left</span> <br>
        <span>
            <input type="button" value="Set All Done"
                    v-on:click="markAllAsCompleted">
            <br>
            <input type="button" value="delete completed"
                    v-on:click="deleteCompleted"
                    v-if="numAllTodos!=0">
            <input type="button" value="delete all"
                    v-on:click="deleteAll"
                    v-if="numAllTodos!=0">
            <br>
            <input type="button" value="show doing"
                    v-on:click="showDoing"
                    v-bind:class="{selected:show==='doing'}"
                    v-if="numAllTodos!=0">
            <input type="button" value="show completed"
                    v-on:click="showCompleted"
                    v-bind:class="{selected:show==='completed'}"
                    v-if="numAllTodos!=0">
            <input type="button" value="show all"
                    v-on:click="showAll"
                    v-bind:class="{selected:show==='all'}"
                    v-if="numAllTodos!=0">
        </span>
    </div>
</div>
<script src="https://cdn.bootcss.com/vue/2.5.16/vue.js"></script>
<script>
    let Id = 0;
    let inputting = false;

    var app = new Vue({
        el: '#todo-app',
        data: function () {
            return{
                todos:[],
                newTodoTitle: '',
                show: 'all'
```

```
            }
        },
        computed:{
            numNotCompleted: function () {
                return this.todos.filter(todo => !todo.completed).length;
            },
            numAllTodos: function () {
                return this.todos.length
            },
            filteredShowTodos: function () {
                if (this.show === 'completed'){
                    return this.todos.filter(todo => todo.completed);
                }else if (this.show ==='doing'){
                    return this.todos.filter(todo => !todo.completed);
                }else {
                    return this.todos;
                }
            }
        },
        methods:{
            addTodo: function () {
                if (this.newTodoTitle.trim()!==''){
                    this.todos.push({id: Id++, title: this.newTodoTitle,
completed: false, buttonEdit: false});
                    this.newTodoTitle = '';
                }
            },
            markAsCompleted: function (todo) {
                todo.completed = true;
            },
            markAsNotCompleted: function (todo) {
                todo.completed = false;
            },
            deleteTodo: function (todo) {
                if (!confirm("Delete this TODO?")) {
                    return
                }
                this.todos.splice(this.todos.indexOf(todo), 1);
            },
            pressEditButton: function (todo) {
                todo.buttonEdit = !todo.buttonEdit;
                this.newTodoTitle = '';
            },
            editTodo: function (todo) {
                if (this.newTodoTitle.trim() !== '') {
                    this.todos.splice(this.todos.indexOf(todo), 1);
                    this.todos.push({id: todo.id, title: this.newTodoTitle,
completed: false, buttonEdit: false});
                    this.newTodoTitle = '';
                }
            },
            markAllAsCompleted: function () {
                this.todos.forEach((item) => {
                    this.markAsCompleted(item)
                });
            },
            deleteCompleted: function () {
                if (confirm("Delete all completed TODOs?")) {
```

```
                    this.todos = this.todos.filter(todo => !todo.completed)
                }
            },
            deleteAll: function () {
                if (confirm("Delete all TODOs?")) {
                    this.todos.forEach((item)=>{
                        this.todos = [];
                    })
                }
            },
            showCompleted: function () {
                this.show = 'completed'
            },
            showDoing: function () {
                this.show = 'doing'
            },
            showAll: function () {
                this.show = 'all'
            }
        },
        directives:{
            focus:{
                inserted: function (el) {
                    el.focus()
                }
            }
        }
    })
</script>

</body>
</html>
```